Fact Oriented Business Service Modeling

Peter Bollen

Department of Organization and Strategy
Faculty of Economics and Business Administration
University of Maastricht

6200 MD Maastricht, the Netherlands
p.bollen@os.unimaas.nl

Abstract. In this paper we will present the results of research into fact-oriented business service modeling. The set of modeling constructs that are defined in this paper are fully ‘compatible’ with the models in the data-oriented perspective in the fact oriented school of conceptual modeling.

Keywords: Business Process Modeling and Improvement, modelling Languages, Business Rules, Process Models.

1 Introduction

Most enterprises today can be considered service providers in one way or other. Many of these service providing enterprises deliver end-products or services to their customers that are the result of some kind of informational activity, e.g. the booking of a holiday at a travel-agency, the preparation of a company’s balance sheet by an independent accountant or the development of a supply chain management system.

Two important schools in the conceptual modeling of informational activity are the (extended) entity relationship (E)E-R approach [1,18] and the fact-oriented approach: NIAM [19] and ORM [6]. Most research in the (E)E-R and fact-oriented approaches has been directed towards the *data-oriented perspective* from the IFIP-CRIS framework [12]. In the eighties a number of system development methodologies were proposed that covered both the data-oriented, process-oriented and behaviour-oriented perspectives [5,11,14]. In the nineties a research school on ‘workflow management’ emerged within the information systems research community (see for a good literature review [15]). Around that time the business process reengineering ‘paradigm’ [3,7] in combination with the increasing popularity of Enterprise Resource Planning packages (e.g. SAP, see [2]), lead to the development of domain-oriented analysis methods of which the ARIS-based Business Process Modeling [16] and BML [8] are examples.

1.1 Related work

In figure 1 we have given the necessary documents for the three perspectives in the IFIP-CRIS architecture (based on [9]). In this paper we will focus on the documents in the middle column of figure 1 that represent the process-oriented perspective. The documents in the left-hand column of figure 1 refer to the data-oriented perspective (an example can be found in [10]). The documents in the right-hand column in figure 1 refer to the behaviour-oriented perspective. The definition of the modeling constructs and methodology for the 'behaviour-oriented' perspective will be subject of another paper. In [9] the union of the meta process model and meta behaviour model is put into the architecure as 'program grammar'.

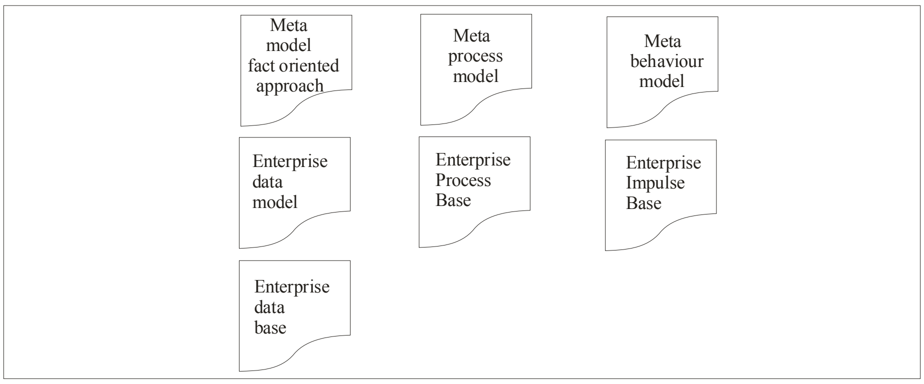

Fig. 1. Documents in the data-, process- and behaviour-oriented perspectives

In analogy to the Conceptual Schema Design Procedure (CSDP) in [10] for the data-oriented perspective, we will give an outline of a 'CSDP' in this paper for the process-oriented perspective, that specifies how a business analyst can create the *enterprise process base* as a declarative representation for the processes in an enterprise subject area. The possible 'process-oriented' models that can exist for the application area and respecting the borders of the application that are imposed by the *Universe of Discourse* (UoD) in the 'data-oriented' perspective.

The remainder of this paper is organized as follows: in section 2 the focus will be on the constructs in the process-oriented perspective, in section 3 the methodology for instantiating these constructs in a specific application area will be given, in section 4 some conclusions will be given.

2 The constructs for the process-oriented perspective

The process perspective in an enterprise subject area is concerned with 'how' fact instances can be composed from other fact instances. In enterprises we can consider facts as either an *outcome* of an enterprise activity or an *ingredient* for an enterprise activity. Enterprise activities are executed under the responsibility of a *user* from a *user group*. We will call a *user* that creates facts, an *active user*. Users that 'consume' instances of fact types in their daily activities are called *passive users*. The border concept in the process perspective will show what *user groups* can be held responsible for the creation of fact instances in the UoD. We will call this border concept: *the*

Sphere of Influence (SoI)[13: 116].

2.1 Definition of conceptual process types

Consider two different examples of billing, for example, in a bistro and in a fast-food restaurant. Although the *spheres of influence* are different in these examples, the description of the informational activity that creates the *order total* on each order receipt in terms of instances of fact types in the application data model is identical (i.e. it is 'organization independent', see [4]). This indicates the need for a theoretical construct that abstracts from the concrete way in which a fact-creating activity is performed (e.g. performed manually by a bistro waiter or automatically under the responsibility of a fast-food restaurant counter employee). This theoretical construct is the *conceptual process instance*.

Definition 1. A *conceptual process instance* is the abstraction of an organizational activity that is responsible for the creation of (a) *fact instance(s)* by an *active user*.

Definition 2. A *conceptual process type* is the intension of a subset of the conceptual process instances that are responsible for the creation of fact instances of the same (set of) fact type(s) by active users in one or more user groups.

2.1.1 Derivation process types

The fact type(s) of the fact instances created in (an) instance(s) of a conceptual process type will be referred to as the *resulting fact type(s)* for the conceptual process type. An (the) *ingredient fact type(s)* of a conceptual process type specifies what the fact type(s) is (are) for the fact instances that serve as an input for the creation of a fact in a process instance of a given conceptual process type.

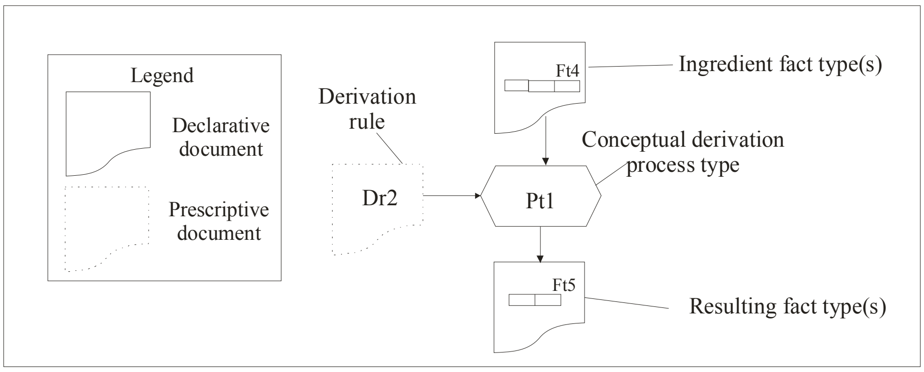

Fig. 2. Conceptual derivation process type

The 'underlying mechanism' that creates fact instance *fact 1* is a function defined on the *ingredient fact instances fact 2, fact 3* and *fact 4.* In case the 'underlying mechanism' is a procedure or a derivation rule that specifies for *all* instances of the conceptual process type how the resulting fact instance(s) can be derived from the ingredient fact instances we will call such a conceptual process type a *derivation process type* (see figure 2).

Definition 3. A *derivation process type* is a conceptual process type whose process instances create fact instances by applying the same derivation rule on instances of the same ingredient fact type(s) (that are contained in the application's data model).

The specification of a *derivation rule* for a given conceptual process type can be considered another semantic bridge in the *process-oriented perspective*. In this specification process the variables in the derivation rules are assigned specific semantics in terms of roles of the *application data model* and the arguments in the *process type argument set* (see section 2.2).

2.1.2 Determination process types

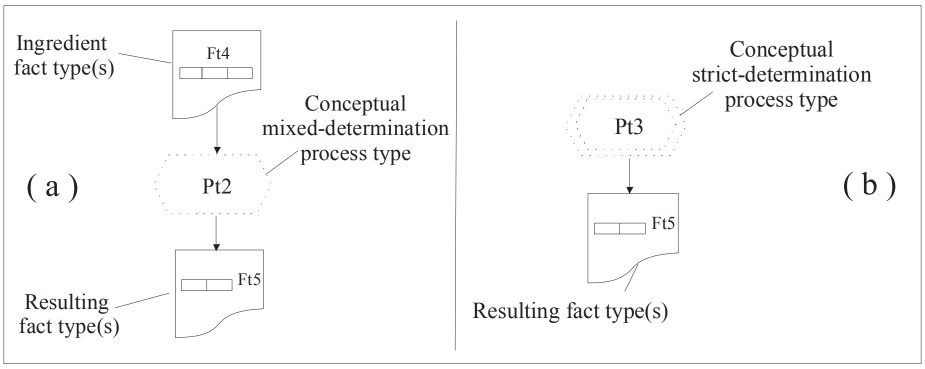

Fig.3. Conceptual determination process types

Some facts will be created without a known (or existing) derivation rule. For example the creation of the Christian name of a new-born. However, in many cases the creation of such a fact is subject to constraints. In the example of the name assignment for a newborn, the following constraint exists: a baby of the female sex must be assigned a girl's name and a baby of the male sex must be assigned a boy's name (eventually from a predefined list of names). We will call the process instances that create these facts, *determination* process instances.

The first group of determination process types is the group of *mixed-determination* process types. The availability of ingredient fact instances is necessary here. However, the *derivation rule* is not known (at least at this moment (see figure 3a)).

Definition 4. A *mixed determination process type* is a conceptual process type in which the active user uses instances of the same ingredient fact types (that are contained in the application's data model) for all process instances.

The conceptual process that creates the names of a newborn baby: *We have decided to call you John. We have decided to call you Alice.* These examples do not involve any derivation rule or (formal) procedure, but it is assumed that ingredient fact instances exist, for example: *John is the name for a boy, Alice is the name for a girl, The child that should be named is a girl* must be known, before a name can be created for a specific child. The way in which a name is assigned in a specific instance, however, can not be determined in advance. Some people might select the name of their own father or mother for their child. Others might choose the name of their favourite rock star. On a 'process type' level, however, we can never know what selection criterion (or derivation rule), will be applied in a specific process instance. The same parent will probably use, if at all, different criteria for every newborn.

In addition to *derivation* and *mixed-determination* process types we can distinguish conceptual process types which have no known and fixed set of ingredient fact type(s) and derivation rules: *strict-determination process types* (see figure 3b). This type of proces is used in managerial decision making, for which, in some cases, *decision support systems* are employed: "The user may only need 40-100 data variables, but they must be the right ones; and what is right may change from day to day and week to week." [17: 21].

Definition 5. A *strict-determination process type* is a conceptual process type in which the active user does not use a known derivation rule all the time and the active user does not use instances of the same ingredient fact types (that are contained in the application's data model) in all process instances.

2.2 The instantiation of conceptual process types.

We now take the *enterprise data base* as a starting point and subsequently apply definitions 4 and 5 that tells us that every fact instance is created in a conceptual process instance. The collection of *conceptual process types* that are relevant for the enterprise subject area are recorded in the *enterprise process base* (see figure 1).

Now we must take the existence of a *conceptual process type* as a starting point and ask ourselves how a *conceptual process type instance* is created. For this instantiation we, generally, need parameters that tell us *what* fact instances will be the 'tangible' end results of the execution of a conceptual process and what other values are needed for such a process execution. We will call such a set of parameters: the *conceptual process type argument* (see figure 4a).

Definition 6. A *conceptual process type argument* specifies the types of values that must be specified for instantiating a conceptual process.

If we consider the derivation process type *create-order-total* in figure 4, it will only create (a) fact instance(s) of fact type FT5 when at least one fact instance of fact type FT4 exists in the *application data base* (see figure 4a) in which the value for the role 'order code' is equal to the value for the process argument 'arg1'. If we inspect the derivation rule for this conceptual process type and the instantiation values for the *process type argument* it should be clear whether the execution of the process will lead to a result **before** the derivation rule is actually executed or fact instance(s) are determined by a active user).

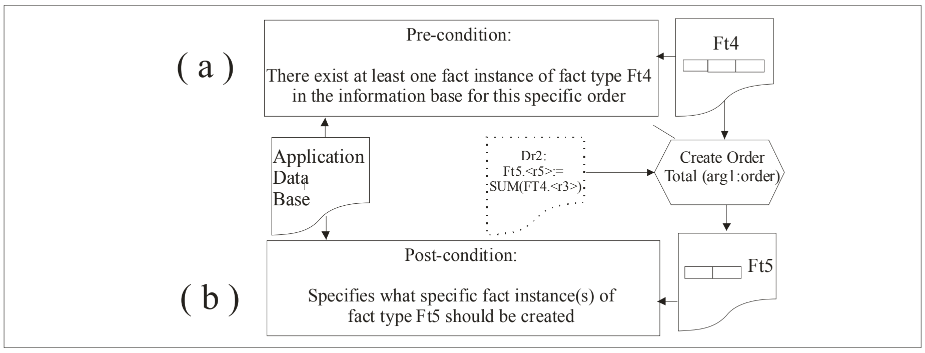
(a)
Pre-condition:
There exist at least one fact instance of fact type Ft4 in the information base for this specific order
Ft4
Application Data Base
Dr2: Ft5.<r5>:= SUM(FT4.<r3>)
Create Order Total (arg1:order)
(b)
Post-condition:
Specifies what specific fact instance(s) of fact type Ft5 should be created
Ft5

Fig. 4. Conceptual process execution: (a) pre-condition, (b) post-condition

The *pre-condition* for a conceptual process type serves as checking mechanism for the instantiation of a process type. If the *pre-condition* is violated by the actual content of the *enterprise data base*, the process will **not** be executed and (a) resulting fact instance(s) will **not** be created.

Definition 7. A *precondition* in a conceptual process type checks whether the required *input fact instances* for the *derivation process* or *the mixed determination process* exists in the *enterprise data base.*

The post-condition specifies what the fact argument is for the facts that will be created in the conceptual process (see figure 4b). Furthermore, it is specified *how* the fact values will be created in the conceptual process will be obtained. In case of a *derivation process* a reference is given to a derivation rule. In case of a *mixed-* or *strict-* determination process, it is stated that (a) fact(s) has (have) to be created (by a active user). This post-condition specifies how the resulting fact type(s) of the process type, must be instantiated as a function of the values for the process argument.

Definition 8. A *post-condition* of a conceptual process type specifies (parts of) the fact argument for the instances of the resulting fact type for the conceptual process. A *post-condition* in a conceptual process indicates that (a) fact value(s) ha(s)ve to be determined. A *post-condition* in a derivation process type specifies what derivation rule is used for the creation of the resulting fact instance(s).

Example 1:

```
P1 create order total<{(arg1,order)}>
IF  there exist an instance of FT4
 SUCH THAT FT4.<r2>=arg1                              {pre-condition}
THEN create an instance of fact type FT5
 SUCH THAT FT5.<r4>:= arg1                            {post condition}
          FT5.<r5>:=DR2
            DR2:= Σ FT4.<r3> [where FT4.<r2>='arg1'] {der.rule}
ENDIF
```

In example 1 we have given a complete specification of the *pre-condition*, *post-condition* and *derivation rule* and how they are related. We will now simplify the specification of a conceptual process type by dividing such a specification in (at most) 3 parts. In the case of a *derivation process type* we will specify a *precondition*, a *postcondition* and a *derivation rule*. In case of a *mixed-determination process type* we will specify the *precondition* and *postcondition* and, finally, in case of a *strict-determination process type* we will only specify the *postcondition*.

3 The modelling methodology for the process-oriented perspective

In order to be able to model the process-oriented features for fact types that are contained in the application's data model but that are created in conceptual process instances that are executed by active users *outside* the focal SoI we need to introduce a fourth conceptual process configuration: the *enter* process type.

Definition 9. An *enter* process type models the process-oriented characteristics for those fact instances of fact types that are contained in the enterprise data model but that are 'created' in conceptual processes by active users *outside* the SoI of the enterprise subject area.

We will illustrate the application of the process modelling constructs using the ABC payroll case study

<u>Example 2: The ABC payroll business service example:</u>

The users in the user groups of the payroll department of branch X of the ABC company, 'decide' how many hours an employee has worked in a given week by inspecting work-order documents and taking additional information into account, e.g. traveling time and information that was obtained in personal contact with the employees. For some employees no work-order documents exist, and therefore the determination of their work-hours is entirely based upon facts that are not contained in the current UoD of the ABC example. The active users in this department furthermore decide upon the gross salaries for the employees that are directly recruited. Although the criteria that determine the salary for each employee are known, the facts that are needed for applying these criteria are not available in the current UoD. The net salary is calculated outside the payroll's enterprise area by a payroll service provider. The *gross-to-net* calculation rules are applied by this outside service-agency, and therefore are not accessible by the active users payroll department of the ABC company. Under some conditions it is possible that the working hours for contractors must be recorded although these contractors are not on the company's payroll. In addition it is possible that employees are on the payroll who are hired under the responsibility of a temping-agency. The users in the user groups of the payroll department of the branch X of the ABC company, are also responsible for knowing the highest (gross) salary for an employee at any time. The SoI consists of the users in the user group of the payroll department of branch X. The content of the fact-oriented data model in figure 9 can be summarized as follows. There exists fact types that declare the existence of a person

(Ft9), that declare that a person earns a gross salary (Ft7), that a person has worked a specific number of hours in a week (Ft8), that there is a highest (gross) salary for an employee (Ft10), and that a person earns a net salary (Ft11). The resulting fact-oriented data model for this example is given in figure 5.

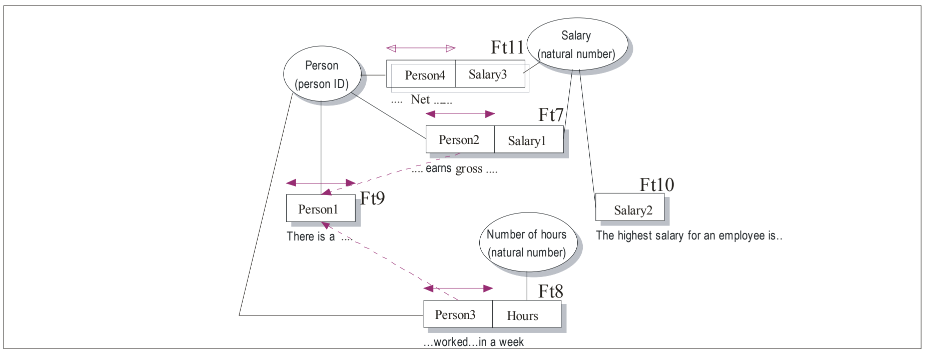

Fig.5. Fact oriented application data model for the payroll example

3.1 A procedure for deriving the process base

In figure 6 we have given a summary of the design procedure for creatuing an application's process base.

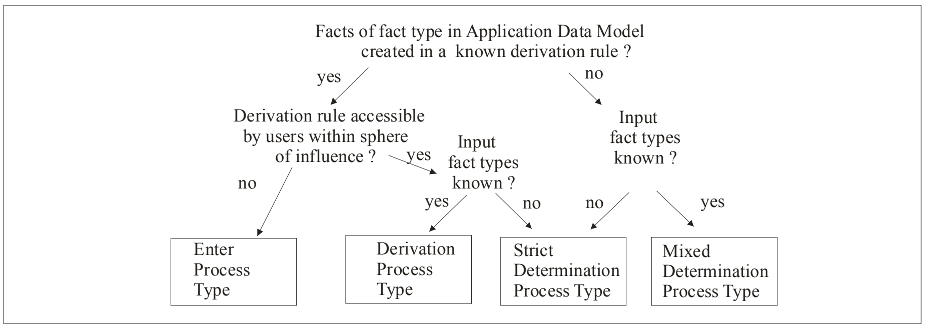

Fig. 6. Procedure for the determination of process type signature for given UoD and SoI

It should be noted that the *enter* process types never have a process type argument, because instances of such a conceptual process type do not have to be instantiated within the *SoI* under consideration.We can now easily derive an application process base for a given UoD and SoI by applying the decision tree from figure 6. The interaction between the UoD (what fact types are relevant for the enterprise subject area) and the SoI (what active users are contained in the enterprise subject area) if not properly managed can be a risk resulting in project delays and project cost overruns in the development life cycle of business information systems. In figure 7 we have given the complete 'as-is'process base for the payroll business service example.

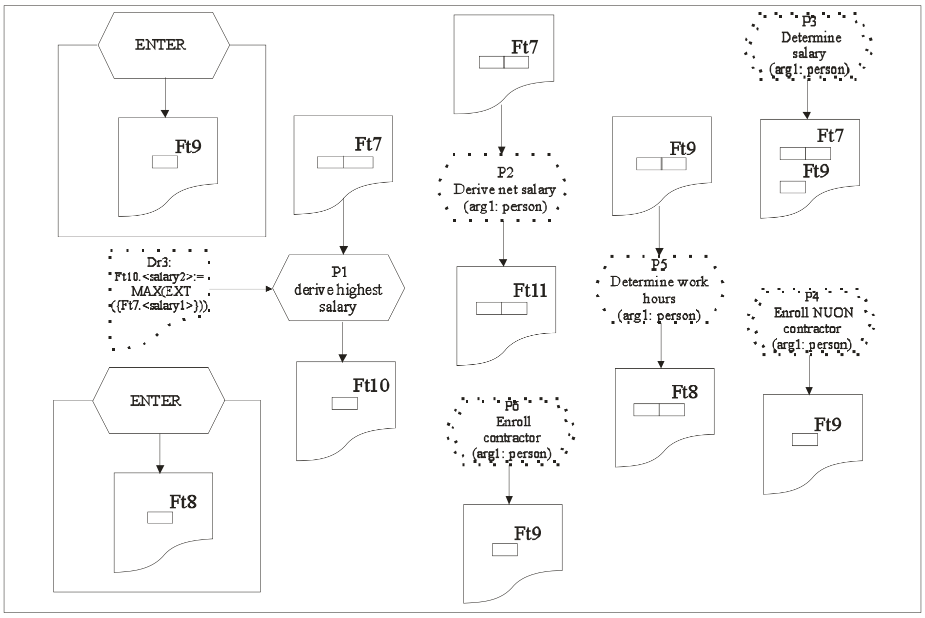

Fig. 7.'As-is' application process base for the payroll business service example

We note that for each fact type from the models in the data-perspective at least one process configuration must be contained in the application's process base. To determine to what process type a process instance belongs, that creates an instance of a fact type (that can be created in 2 or more process types), we need an enterprise impulse base, that specifies under what conditions a *specific* process type will be instantiated to create an instance of such a fact type.

4 Conclusions

In this paper we have derived the modeling constructs and an accompanying methodology for the creation of a process base for a given subject area. The constructs that were introduced in section 2 of this paper allow us to describe the extent as to which organizations have discretion with respect to the fact generating activities within the *SoI*. The definition of three different conceptual process types in combination with the process border-concept of Sphere of Influence (SoI) has resulted in the existence of 4 conceptual process configurations for a given enterprise subject area with a known *UoD* and a known *SoI*. The ability to model conceptual knowledge processes that have a 'tacit' nature and the extent in which the 'codifiable' properties of these tacit knowledge processes can be modeled makes the constructs in the meta process model in this paper applicable in service enterprises .The modeling constructs also allow us to model every type of decision process in terms of its equivocality and uncertainty. In the context of creating conceptual models in the early stages of the Systems Development Life Cycle (SDLC), the aforementioned constructs and methodology can be used as well. The resulting process models can be easily mapped onto application programs that work on an application data base, by mapping the derivation process types in a straightforward manner.

References

1. Chen, P.: The Entity-Relationship model: Towards a unified view of data. ACM TODS **1** (1) (1976) 9-36
2. Curran, T., Ladd, A.: SAP R/3 business blueprint. Prentice-Hall (2000)
3. Davenport,T., Short, J. :The new industrial engineering: Information Technology and Business Process Redesign. Sloan management Review. Summer (1990): 11- 27.
4. Gorry,G., Scott Morton, M.: A framework for management information systems. Sloan Management Review, Fall (1971)
5. Gustafsson,M., Karlsson, T., Bubenko J.: A declarative approach to conceptual information modeling, in: W.Olle, H.Sol and A. Verrijn-Stuart (eds.),Information System Design Methodologies- a comparative review, North-Holland, (1982) 93-142.
6. Halpin, T.: Information Modeling and Relational Databases, Morgan Kaufmann . (2001)
7. Hammer, M.: Reengineering work: Don't automate,obliterate. HBR.july(1990)104-112.
8. Johannesson, P., Perjons, E.: Design principles for process modelling in enterprise application integration. Information Systems **26** (2001): 165-184
9. Nijssen,G.: An Axiom and Architecture for Information Systems. In*:* Falkenberg, E., Lindgreen, P. (eds.): Information System Concepts,North-Holland, (1989) 157- 175.
10. Nijssen, G., Halpin, T.: Conceptual schema and relational database design. Prentice-Hall, Englewood Cliffs (1989).
11. Olivé, A.:Dades- a methodology for specification and design of information systems design and management. in: Olle et. al. (eds.), Information System Design Methodologies- a comparative review, North-Holland, (1982) 285-334.
12. Olle, T.W., J.Hagelstein, I.G. Macdonald, C. Rolland, H.G. Sol, F.J.M. Van Asche and A.A. Verrijn-Stuart. Information Systems Methodologies- A Framework for Understanding, North-Holland (1988).
13. Parker, M.: Enterprise information-analysis: Cost-benefit analysis and the data-managed system. IBM systems journal, 21(1) (1982) 108-123.
14. Rolland, C., Richard, C.:The REMORA Methodology for Information System Design and Management. Information System Design Methodologies- a comparative review (1982)
15. Salimifard, K., Wright, M.: Theory and Methodology: Petri net-based modelling of workflow systems . European Journal of Operations Research **134** (2001) 664-676
16. Scheer, A.: ARIS-Business Process Modeling, 2nd edition, Springer, Berlin (1999)
17. Sprague Jr., R. : A Framework for the Development of Decision Support Systems. MIS Quarterly. December (1980)
18. Teory, T., Yang, D., Fry, J.:A logical design methodology for relational databases using the extended E-R model. ACM Computing Surveys, **18**(2) (1986):197-222
19. Verheijen,G., van Bekkum J.: NIAM: An Information Analysis Method. In: Verrijn-Stuart,A., Olle T., Sol H., (eds.): proceedings CRIS- 1, North-Holland (1982) 537-590.